Time versus space trade-offs for rendezvous in trees

Jurek Czyzowicz, Adrian Kosowski, Andrzej Pelc

Time vs. space trade-offs for rendezvous in trees*

Jurek Czyzowicz† Adrian Kosowski‡ Andrzej Pelc§

Abstract

Two identical (anonymous) mobile agents start from arbitrary nodes of an unknown tree and have to meet at some node. Agents move in synchronous rounds: in each round an agent can either stay at the current node or move to one of its neighbors. We consider deterministic algorithms for this rendezvous task. The main result of this paper is a tight trade-off between the optimal time of completing rendezvous and the size of memory of the agents. For agents with k memory bits, we show that optimal rendezvous time is $\Theta(n+n^2/k)$ in n-node trees. More precisely, if $k \geq c \log n$, for some constant c, we design agents accomplishing rendezvous in arbitrary trees of size n (unknown to the agents) in time $O(n + n^2/k)$, starting with arbitrary delay. We also show that no pair of agents can accomplish rendezvous in time $o(n+n^2/k)$, even in the class of lines of known length and even with simultaneous start. Finally, we prove that at least logarithmic memory is necessary for rendezvous, even for agents starting simultaneously in a n-node line.

Keywords: rendezvous, anonymous agents, time, memory space.

1 Introduction

Two identical mobile agents, starting at two nodes of a network, have to meet in the same node at the same time. Agents move along links from node to node, in synchronous rounds: in each round an agent can either stay at the current node or move to one of its neighbors. This task is known as rendezvous [1, 4], and its various applications are discussed in [3]. Rendezvous has applications even in human interaction, e.g., when agents are people that have to meet in a city whose streets form a network, or when rescuers must find a lost tourist in the mountains. In computer science applications, mobile agents usually represent software agents in computer networks, or mobile robots, if the network is a labyrinth or is composed of corridors in a building. The reason to meet may be to exchange data previously collected by the agents, or to plan some future task, such as periodic network maintenance or sharing a computational task to speed it up.

The network is modeled as an undirected connected graph. We make three assumptions, standard in the literature on rendezvous in networks. The first is that nodes of the network are unlabeled. In other words, we seek rendezvous algorithms for agents that do not rely on the knowledge of node labels, and can

*A preliminary version of this paper appeared in the Proc. 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2012), June 2012, Pittsburgh, USA, 1-10.

†Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada. E-mail: jurek@uqo.ca. Supported in part by NSERC discovery grant.

‡CEPAGE Project, Inria Bordeaux Sud-Ouest, 33400 Talence, France. E-mail: kosowski@labri.fr. This research was partially done while Adrian Kosowski was working at the Department of Algorithms and System Modeling of the Gdańsk University of Technology and during his visit to the Research Chair in Distributed Computing of the Université du Québec en Outaouais. Supported in part by ANR project DISPLEXITY and by NCN under contract DEC-2011/02/A/ST6/00201.

§Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada. E-mail: pelc@uqo.ca. Supported in part by NSERC discovery grant and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

work in anonymous graphs as well (cf. [3]). The importance of designing such algorithms is motivated by the fact that, even when nodes are equipped with distinct labels, agents may be unable to perceive them because of limited sensory capabilities, the labels of the nodes may change in time, or nodes may refuse to reveal their labels, e.g., due to security or privacy reasons. Specific examples of agents unable to perceive their environment may include sandboxed mobile agents, such as script applications or applets running in a web-browser with no access to persistent information which would uniquely identify their host over a long period of time, or untrusted applications running on a smartphone with restricted privileges. In physical scenarios, one can imagine robots moving around a terrain or maze, whose sensory capabilities involve touch or obstacle detection, and whose capabilities of vision are extremely limited.

The second assumption is that edges incident to a node v have distinct integer labels. Every undirected edge $\{u, v\}$ has two labels, which are called its *port numbers* at u and at v. The port numbering is *local*, i.e., there is no relation between port numbers at u and at v. Note that, in the absence of port numbers, the adversary could prevent rendezvous by always avoiding to direct an agent to some edge incident to the current node. The third assumption is that agents cannot leave any marks on visited nodes. While both this assumption [16, 32] and the opposite one, i.e., allowing agents to leave either tokens [11, 27] or larger messages on whiteboards at nodes [34], have been considered in the literature, the advantage of designing rendezvous algorithms not relying on marks is two-fold. On the one hand, nodes may not be equipped with such whiteboards designated to leave marks, and on the other hand, nodes need not be cooperative and may erase or alter the messages left by the agents after their visit.

In this paper we focus our attention on deterministic rendezvous in trees, establishing tight trade-offs between the optimal time of completing rendezvous and the size of memory of the agents. It should be noted that such a goal for arbitrary graphs seems to be presently out of reach, as even the optimal time of rendezvous with unlimited memory is not known, and the existing upper bounds on rendezvous time in arbitrary graphs [16, 25, 32] are large polynomials that seem far from optimal. Generalizing our result for trees (tight time-memory trade-offs) to arbitrary graphs would imply the solution of the above open problem.

It is well known that deterministic rendezvous with simultaneous start is impossible if the initial positions of the two agents are symmetric, i.e., if there is a port-preserving automorphism of the tree that carries one node on the other. (Indeed, in this case the positions of agents at each round will remain symmetric, thus precluding rendezvous.) Hence we always assume that the initial positions of agents are not symmetric.

1.1 Our results

The main result of this paper is a tight trade-off between optimal time of completing rendezvous in a tree and the size of memory of the agents. The agents do not know the topology of the tree or any bound on its size. For agents with k memory bits, we show that optimal rendezvous time is $\Theta(n+n^2/k)$ in n-node trees. More precisely, if $k \geq c \log n$, for some constant c, we show agents accomplishing rendezvous in arbitrary trees of unknown size n in time $O(n + n^2/k)$, starting with arbitrary delay. Our algorithm works for all trees and configurations of agents for which rendezvous is always possible, regardless of the starting delay. The approach makes use of an adaptation of Duval's efficient algorithm for finding the maximum suffix of a string [17] in order to compute a form of "signature" (label) for each of the agents, which is sufficient to break symmetry, in optimal time given the agents' memory. Once symmetry has been broken by the agents, we show that rendezvous can be achieved in time of the same order as that required to construct the signatures of the agents. Designing such a time-optimal algorithm proves particularly complicated for agents with memory between $\Theta(n/\log n)$ and $\Theta(n)$, for which we have to provide a separate efficient algorithm for locating the rendezvous point. This is based on a specific procedure of "trimming" the tree in order to find its central node or edge quickly. We start by presenting our algorithms for the case of trees of maximum degree 3 and approximately known size, and then proceed to lift both of these assumptions.

We also show that no pair of agents can accomplish rendezvous in time $o(n + n^2/k)$, even in the class of lines and even with simultaneous start. Finally we prove that, contrary to an erroneous result established in [21], a logarithmic number of bits of memory are needed for rendezvous, even for agents starting simultaneously in a n-node line, for arbitrarily large n.[1]

1.2 Related work

The literature on rendezvous can be divided into two currents, significantly differing in the methodology and algorithm construction. The first concerns randomized rendezvous, where the initial positions of agents are random with some given distribution over the environment, and/or the rendezvous algorithm contains randomized inputs (coin tosses). An extensive survey of randomized rendezvous in various scenarios can be found in [3], cf. also [1, 2, 6, 10, 24]. Several authors considered randomized rendezvous time in the geometric scenario (rendezvous in an interval of the real line, see, e.g., [10, 11, 23], or in the plane, see, e.g., [7, 8]). Memory needed for randomized rendezvous in the ring is discussed, e.g., in [26].

For the deterministic setting a lot of effort has been dedicated to the study of the feasibility of rendezvous, and of the time required to achieve this task, when feasible. For instance, deterministic rendezvous with agents equipped with tokens used to mark nodes was considered, e.g., in [27]. Deterministic rendezvous of agents with unique labels was discussed in [15, 16, 25, 32]. (In the latter scenario, symmetry is broken by the use of the different labels of agents, and thus rendezvous is sometimes possible even for symmetric initial positions of the agents). The algorithm of Dessmark et al.[16] is deterministic and relies on repeated traversals of the graph, in a way modulated by the label of the agent, to achieve rendezvous in polynomial time. It is assumed that the system operates in synchronous rounds, but that one of the agents may be delayed and may only appear in the graph after some period of time τ, and that the time required for rendezvous is only counted from the moment of appearance of the later agent. The runtime of their algorithm is given as $\tilde{O}(n^5\sqrt{\tau l} + n^{10}l)$, where n is the number of nodes of the graph, and l represents the bit length of the shorter of the agents' labels. Subsequently, Kowalski and Malinowski [25] and Ta-Shma and Zwick [32] designed algorithms with runtime which is independent of the delay τ, namely, $\tilde{O}(n^{15} + l^3)$ and $\tilde{O}(n^3d^2l)$, respectively, where d is the maximum degree of the graph. Interestingly, all of these algorithms operate without knowledge of an upper bound on n.

Memory required by the agents to achieve deterministic rendezvous has been studied in [20, 21] for trees and in [13] for general graphs. In [13] it is shown that the minimum memory size for rendezvous in arbitrary n-node graphs is $\Theta(\log n)$. It should be noted that, unlike the present paper, papers [13, 20, 21] discussed only memory size without caring about time of rendezvous. The work [13] relies on Universal Exploration Sequences for graphs of size n^2 to compute distinct labels for two agents occupying non-symmetric positions in a graph of order n. The time of rendezvous which can be directly inferred from [13] for general graphs is $\tilde{O}(n^{15})$, up to polylogarithmic factors. For the papers on trees [20, 21], the time of rendezvous with logarithmic size of memory is of course polynomial, though the construction of the algorithm necessitates at least n^3 steps in some scenarios. These works also rely on computing distinguishing labels for the agents, but by virtue of the tree topology these labels can be somewhat shorter than for the case of general graphs considered in [13]. Even so, the approach which we present in this paper is noticeably faster.

Rendezvous time (both deterministic and randomized) of anonymous agents in trees without marking nodes has also been very recently studied in [18]. It was shown therein that deterministic rendezvous in n-node trees can be always achieved in time $O(n)$, but only when the memory size of the agents is at least linear. (This can be seen as one extremal point of the tradeoff we are considering in this paper.)

We remark that the case of trees is also special for the exploration problem: whereas exploring all

[1] A result from [21] implies that rendezvous with simultaneous start from arbitrary non-symmetric initial positions in a n-node line is possible with $O(\log\log n)$ bits of memory. This result from [21] is untrue, although it holds in the model with adversarial port labelings, cf. the corrected version of that paper [22].

nodes of the graph with termination and without knowledge of an upper bound on the number of nodes is impossible in general (even when the graph is a ring), the problem can be solved in trees, even using agents with $O(\log n)$ memory [5].

A natural extension of the rendezvous problem is that of gathering [19, 24, 28, 33], when more than two agents have to meet in one location. In [34] the authors considered rendezvous of many agents with unique labels.

Apart from the synchronous model used in this paper, several authors have investigated asynchronous rendezvous in the plane [12, 19] and in network environments [9, 14, 15]. In the latter scenario the agent chooses the edge which it decides to traverse but the adversary controls the speed of the agent. Under this assumption rendezvous in a node cannot be guaranteed even in very simple graphs, hence the rendezvous requirement is relaxed to permit the agents to meet inside an edge. In [9] the authors study the memory size needed for time-optimal asynchronous rendezvous in trees. (They do not allow rendezvous inside an edge, but for symmetric trees they allow that agents terminate not in one node but in two adjacent nodes.) They show that the minimum number of memory bits to achieve rendezvous in linear time is $\Theta(n)$.

For a more extensive survey of models and approaches for the rendezvous problem, we refer the reader to the survey paper [30].

2 Framework and Preliminaries

We consider trees with unlabeled nodes and labeled ports. The port numbers at a node of degree d are $0, 1, \ldots, d-1$. The number of nodes of a tree is called its *order*. An isomorphism between trees $T = (V, E)$ and $T' = (V', E')$ is a bijection $f : V \to V'$, such that for any $u, v \in V$, u is adjacent to v if and only if $f(u)$ is adjacent to $f(v)$. An isomorphism is said to *preserve port numbering* if for any $u, v \in V$, the port number corresponding to edge $\{u, v\}$ at node u is equal to the port number corresponding to edge $\{f(u), f(v)\}$ at node $f(u)$. An automorphism is an isomorphism of a tree on itself. A pair of distinct nodes u, v of a tree is called *symmetric*, if there exists an automorphism f preserving port numbering, and such that $f(u) = v$.

We consider identical mobile agents traveling in trees with locally labeled ports. Unless stated otherwise, the tree and its size are a priori unknown to the agents. We first define precisely an individual agent. An agent is an abstract state machine $\mathcal{A} = (S, \pi, \lambda, s_0)$, where S is a set of states among which there is a specified state s_0 called the *initial* state, $\pi : S \times \mathbb{Z}^2 \to S$, and $\lambda : S \to \mathbb{Z}$. Initially the agent is at some node u_0 in the initial state $s_0 \in S$. The agent performs actions in rounds measured by its internal clock. Each action can be either a move to an adjacent node or a null move resulting in remaining in the currently occupied node. State s_0 determines a natural number $\lambda(s_0)$. If $\lambda(s_0) = -1$ then the agent makes a null move (i.e., remains at u_0). If $\lambda(s_0) \geq 0$ then the agent leaves u_0 by port $\lambda(s_0)$ modulo the degree of u_0. When incoming to a node v in state $s \in S$, the behavior of the agent is as follows. It reads the number i of the port through which it entered v and the degree d of v. The pair $(i, d) \in \mathbb{Z}^2$ is an input symbol that causes the transition from state s to state $s' = \pi(s, (i, d))$. If the previous move of the agent was null, (i.e., the agent stayed at node v in state s) then the pair $(-1, d) \in \mathbb{Z}^2$ is the input symbol read by the agent, that causes the transition from state s to state $s' = \pi(s, (-1, d))$. In both cases s' determines an integer $\lambda(s')$, which is either -1, in which case the agent makes a null move, or a non-negative integer indicating a port number by which the agent leaves v (this port is $\lambda(s') \bmod d$). The agent continues moving in this way, possibly infinitely. The memory of the agent is measured by the number of states of the corresponding state machine, or equivalently by the number of bits on which a state can be encoded. A state machine with K states requires $\Theta(\log K)$ bits of memory.

Since we consider the rendezvous problem for identical agents, we assume that agents are copies A_1 and A_2 of the same abstract state machine $\mathcal{A}$, starting at two distinct non-symmetric nodes v_1 and v_2, called the *initial positions*. We will refer to such identical machines as a *pair of agents*. Hence a pair of

agents executes an identical algorithm. It is assumed that the internal clocks of a pair of agents tick at the same rate. The clock of each agent starts when the agent starts executing its actions. Agents start from their initial positions with *delay* $\theta \geq 0$ between their starting rounds, controlled by an adversary. This means that the later agent appears at its starting position and starts executing its actions θ rounds after the first agent. Agents do not know which of them is first and what is the value of θ. The time of a rendezvous algorithm is the number of rounds since the start of the later agent until rendezvous.

We say that a pair of agents using a deterministic algorithm solves the rendezvous problem *with arbitrary delay* (resp. *with simultaneous start*) in a class of trees, if, for any tree in this class and for any initial positions that are not symmetric, both agents are eventually in the same node of the tree in the same round, regardless of the starting rounds of the agents (resp. provided that they start in the same round).

Consider any tree T and the following sequence of trees constructed recursively: $T_0 = T$, and T_{i+1} is the tree obtained from T_i by removing all its leaves. We define $\hat{T}$ as T_j for the smallest j for which T_j has at most two nodes. If $\hat{T}$ has one node, then this node is called the *central node* of T. If $\hat{T}$ has two nodes, then the edge joining them is called the *central edge* of T. A tree T with port labels is called *symmetric*, if there exists a non-trivial automorphism f of the tree (i.e., an automorphism f such that $f(u) \neq u$, for some $u \in V$) which preserves port numbering. If a tree with port labels has a central node, then it cannot be symmetric. In a non-symmetric tree, every pair of nodes is non-symmetric, hence rendezvous is feasible for all initial positions of agents.

A *basic walk* in a n-node tree T, starting from node v is a traversal of all edges of the tree ending at the starting node v and defined as follows. Node v is left by port 0; whenever the walk enters a node by port i, it leaves it by port $(i + 1)$ mod d, where d is the degree of the node. We sometimes consider more than $2(n - 1)$ steps of a basic walk, noting that this traversal is periodic with a period of length $2(n - 1)$. The basic walk starting at a node v may be uniquely coded by the sequence (string of symbols) $BW(v) = (p_1(v), q_1(v), p_2(v), q_2(v), \ldots, p_{2(n-1)}(v), q_{2(n-1)}(v))$, where $p_1(v) = 0$, $p_i(v)$ is the port number by which the node is left in the i-th step of the walk, and $q_i(v)$ is the port number by which the node is entered in the i-th step of the walk. A pair of nodes v_1 and v_2 of a tree T is not symmetric if and only if $BW(v_1) \neq BW(v_2)$. Thus an agent starting at node v can be uniquely identified in the tree using the string $BW(v)$, or using any string describing a longer traversal which has $BW(v)$ as its prefix. Note that the length of $BW(v)$ is fixed as $2(n-1)$, and the definition of the string $BW(v)$ is completely independent on the upper bound N on n which is known to the agent.

A *reverse basic walk* starting from node w with port p is a traversal of all edges of the tree ending at the starting node w and defined as follows. Node v is left by port p; when the walk enters a node by port i, it leaves it by port $(i - 1)$ mod d, where d is the degree of the node.

For a string σ of length m, the rotation $rot_l(\sigma)$ is the string σ', such that $\sigma'[i] = \sigma[(i + l) \bmod m]$, for all indices $0 \leq i \leq m - 1$. Any string σ can be uniquely encoded by its lexicographically minimal rotation $LMR(\sigma)$ and the smallest non-negative integer l such that $LMR(\sigma) = rot_l(\sigma)$.

3 The rendezvous algorithm

Our presentation of the rendezvous algorithm is divided into three stages. In the first stage we make two simplifying assumptions: (i) we assume that the maximum degree Δ of the tree is bounded by 3; (ii) we assume that the agents know *a priori* some upper bound N on the order n of the tree, such that $N \geq n \geq N/16$. (The reason for the choice of the constant 16 will become apparent in Section 3.3) In the second stage we remove assumption (i), still keeping (ii), and in the third stage we remove both assumptions, thus presenting the general algorithm.

3.1 Trees of approximately known size and bounded degrees

In this section we present the rendezvous algorithm for trees of maximum degree bounded by 3 and assuming that agents know an integer N, such that $N \geq n \geq N/16$. The overview of the algorithm is the following. In the first phase, whose time is $O(n + n^2/k)$, each agent computes an integer value $l \in \{0, 1, \ldots, n-1\}$ called its *signature*, such that agents with non-symmetric initial positions have different signatures. These signatures are used in the second phase to break symmetry and achieve rendezvous. The way in which this is done depends on the amount of memory available to the agents. If the agents have large memory (at least $\Omega(n/\log n)$ memory bits), then they can quickly locate either the central node or the central edge of a specifically chosen subtree of the tree in which they operate. In the first case they meet at its central node, in the second case they use the signatures to meet at one of the endpoints of its central edge. In the case of small memory ($o(n/\log n)$ memory bits), each agent uses a sequence of active and passive periods, each of length $4(N-1)$, determined by the successive bits of its signature: in an active period (bit 1 of the signature) an agent visits all nodes of the tree, in a passive period (bit 0 of the signature) it waits. This guarantees rendezvous in additional time at least $O(n \log n)$ which is dominated by $O(n^2/k)$, for small memory.

Procedure for computing agent signatures

Due to the periodic nature of tree traversal using the basic walk, all the strings $BW(v)$, for $v \in V$, are identical up to rotation, and hence have the same string describing their lexicographically minimal rotation. We define the signature $sig(v)$ of an agent with initial starting position v as the minimum l such that $LMR(BW(v)) = rot_l(BW(v))$. Hence, agents with non-symmetric initial positions have different signatures. Observe that $0 \leq sig(v) < 2(n-1)$, since $BW(v)$ is periodic with a period of length $2n-1$.

To compute the value of $sig(v)$, we apply the following procedure, called FindSignature, which allows an agent starting at node v to detect the starting position of $LMR(BW(v))$ as a rotation of $BW(v)$. To do this, in the pseudocode below we describe a variant of Duval's efficient maximum suffix algorithm [17] (cf. also [31] for an external I/O memory implementation), adapting it for the mobile agent computational model with limited memory. This is, to our knowledge, the first application of Duval's approach in mobile agent computing. Intuitively, the agent makes use of two pointers to symbols of $BW(v)$, represented by positions $left$ and $right$, which it sweeps from left to right. Index $left$ represents the starting position of the lexicographically minimal rotation which has been detected so far, while index $right$ represents the currently considered candidate for such a starting position. Our implementation of FindSignature has two important features. Firstly, the comparison of characters of the string $BW(v)$ is encapsulated in subroutine CompareString $(left, right, maxLength)$, which lexicographically compares the two substrings of $BW(v)$ having length $maxLength$ and starting at offsets $left$ and $right$, respectively. Secondly, the agent is not assumed to know the exact length $4(n-1)$ of sequence $BW(v)$; instead, the known upper bound of $4(N-1)$ is used, without affecting the correctness of the algorithm.

```
procedure FindSignature ()
  left ← 1; right ← 2;
  repeat
    maxLength ← right − left;
    CompareString (left, right, maxLength);
    if 'left' string is greater { at some index } then
      left ← right; right ← right + 1;
    else
      if 'left' string is smaller at index i then
        right ← right + i;
      else {strings are equal}
        right ← right + maxLength;
  until right > 4(N − 1);
  return left;
```

When defining procedure CompareString, we will assume that the agent is equipped with four memory blocks, called *views*, each of which can store a substring of μ successive symbols from the string $BW(v)$, where μ is some integer smaller than $k/4$ (recall that k is the number of bits of memory of the agent).

These views are assumed to have identifiers named $view\_Left$, $view\_Right$, $view\_TempLeft$, and $view\_TempRight$. Procedure CompareString makes use of the views so that the amortized time of comparing μ symbols is $O(n)$. When comparing the first $\mu/2$ symbols of the strings within CompareString, the views $view\_Left$ and $view\_Right$ are used. After that, further use of these views might result in the necessity to shift the position of the views back to the left during some later string comparison. To avoid this, the buffers $view\_TempLeft$ and $view\_TempRight$ are activated at this point.

The auxiliary procedure GetSymbol (pos, $view\_ID$, $newStartPos$) is defined so as to return the (pos)-th symbol of the string $BW(v)$, retrieving its value from the view with the specified identifier $view\_ID$. In the case when index pos is outside the range currently stored in the view, the range of the view is reset to the following: $[newStartPos, newStartPos + \mu - 1]$ within the subroutine UpdateView. The arguments passed to each call of GetSymbol are always such that $pos \in [newStartPos, newStartPos + \mu - 1]$, hence the ($pos$)-th symbol of the string $BW(v)$ may subsequently be returned by GetSymbol using the updated $view\_ID$.

```
procedure CompareString (left, right, maxLength)
  p ← 0;
  repeat
    if p ≤ μ/2 then
      l ← GetSymbol (left + p, view_Left, left);
      r ← GetSymbol (right + p, view_Right, right);
    else
      l ← GetSymbol (left + p, view_TempLeft, left + p);
      r ← GetSymbol (right + p, view_TempRight, right + p);
    if l < r then
      return 'left' string is smaller at index p;
    if l > r then
      return 'left' string is greater at index p;
    p ← p + 1;
  until p = maxLength;
  return strings are equal;

procedure GetSymbol (pos, view_ID, newStartPos)
  if pos not in range stored in view_ID then
    UpdateView (view_ID, newStartPos);
  return port at step pos of basic walk using view_ID;

procedure UpdateView (view_ID, newStartPos)
  perform 4(N − 1) steps of the basic walk and, during the next μ rounds
  starting from the (newStartPos)-th node of the basic walk, store the visited
  ports to view_ID;
  set range of view_ID to [newStartPos .. (newStartPos + μ − 1)];
  perform 4(N − 1) steps of the reverse basic walk, returning to the starting
  node;
```

Lemma 3.1 *For any value of μ, $1 \leq \mu \leq N$, the total number of calls to procedure* UpdateView *in an execution of procedure* FindSignature *is bounded by $48N/\mu$.*

Proof. We first remark that throughout the execution of procedure FindSignature, the values of variables $left$ and $right$ are non-decreasing, and $1 \leq left \leq right \leq 4(N-1)$. Moreover, let c be the total number of calls to procedure CompareString, and let $p(i)$ be the final value of the counter p in the i-th call to this procedure. Then, from the analysis of the duration of the loops of the algorithm [17] it follows that $\sum_{i=1}^{c} p(i) \leq 8(N-1)$.

The proof proceeds by showing that procedure UpdateView is called at most $8N/\mu$ times for each of the views $view\_Left$, $view\_Right$, and at most $16N/\mu$ times for each of the views $view\_TempLeft$, $view\_TempRight$.

First, consider the view $view\_Left$. Suppose that UpdateView was called for this view for some value of variable $left = left_1$. Then, the range of $view\_Left$ is set to $[left_1, left_1 + \mu - 1]$. Observe that the next update of $view\_Left$ in procedure CompareString may only occur when $left + p \geq left_1 + \mu$ for some value of $p \leq \mu/2$, hence, $left \geq left_1 + \mu/2$. Since $1 \leq left \leq 4(N-1)$, the total number of updates of $view\_Left$ can be bounded by $8N/\mu$. The same argument may be used to bound the number of updates of $view\_Right$ by $8N/\mu$.

Now, consider the number of updates of $view\_tempLeft$ in the i-th execution of procedure CompareString. If $p(i) \leq \mu/2$, then $view\_tempLeft$ will not be updated. Otherwise, if an update of the view

occurred for some value of $p = p_1$, then the range of the view is reset to $[left + p_1, left + p_1 + \mu - 1]$. Within this execution of procedure CompareString, the next update of the view may only occur when $p \geq p_1 + \mu$. Consequently, the total number of view updates within the i-th execution of CompareString can be bounded by $\lceil (p(i) - \mu/2)/\mu \rceil \leq 2p(i)/\mu$. Recalling the relation $\sum_{i=1}^{c} p(i) \leq 8(N-1)$, the total number of updates to $view\_tempLeft$ is bounded by $16N/\mu$. The same bound holds for $view\_tempRight$. □

Corollary 3.1 *For any upper bound N, such that $N \geq n \geq N/16$, and $k \geq c \log N$, where c is a constant, an agent starting at node v and equipped with k bits of memory can compute its signature $sig(v)$ in $O(n^2/k)$ rounds by following procedure* FindSignature *for the value $\mu = k/8$.*

Proof. An agent performing procedure FindSignature uses only one variable of $O(\log N)$ bits to store the current position of the agent along the basic walk with respect to its starting node, a constant number of auxiliary variables of size $O(\log N)$ in procedure FindSignature and its subroutines, and 4 views with a range of $\mu = k/8$ each. Each of these views can be implemented by storing an array of μ numbers from the set $\{0, 1, 2\}$, describing the ports of the basic walk, and one number of size $O(\log N)$ describing the starting position of the range stored in the buffer. Hence, the procedure can be performed by an agent with k bits of memory.

In order to bound the number of rounds required for execution, observe that all the computations of the agent may be performed locally, except for the moves of the agent encapsulated in the subroutine UpdateView. Notice that, since the agent never waits in the execution of the procedure, the number of rounds is equal to the number of moves. The number of rounds required to perform the procedure UpdateView is determined by the duration of the $4(N-1)$ steps of the basic walk and the duration of the $4(N-1)$ steps of the reverse basic walk, thus precisely equal to $8(N-1)$. Since the number of calls to UpdateView is bounded by $48N/\mu = 384N/k$ by Lemma 3.1, the number of rounds used by procedure FindSignature is bounded by $3072N^2/k$.

Finally, the correctness of the computation of $sig(v)$ follows from the analysis of the circular string canonization algorithm (cf. [17, 31]) implemented in procedure FindSignature. □

Rendezvous procedure for agents with small memory

The rendezvous procedure for an agent with an already computed signature $sig(v)$ depends on the relation between the number k of memory bits and the known upper bound N on the order of the tree. The first procedure, called SmallMemoryRV, guarantees rendezvous of agents with known signatures in $O(N \log N)$ rounds and using $\Theta(\log N)$ bits of memory. Consequently, the procedure will be applied in the case when $k < N/\log N$, since then $N \log N \in O(N^2/k)$ and the bound of $O(N^2/k)$ on execution time is achieved. A faster procedure for agents with larger memory will be presented further on.

Procedure SmallMemoryRV assigns to each agent a unique label defined as a string of $\lceil 2 \log N + 3 \rceil$ bits, encoding the binary representation of the integer $2sig(v) + 1$. The procedure is composed of phases such that in the i-th phase, depending on the value of the i-th bit of this label, the agent either visits all the nodes of the tree at least twice, or waits at its initial location for a number of rounds corresponding to such an exploration. This is iterated for all i, $1 \leq i \leq \lceil 2 \log N + 3 \rceil$, and then the whole process is repeated until rendezvous is achieved. The traversal of the tree, which needs to be performed as a subroutine, is implemented by $2(N-1)$ steps of the basic walk, and then returning to the starting node in $2(N-1)$ steps of the reverse basic walk.

```
procedure SmallMemoryRV ()
  sig ← FindSignature ();
  repeat
    Oscillate (sig, 2(N − 1), 0);
  until rendezvous;

procedure Oscillate (sig, distance, firstPort)
  for i ← 1 .. ⌈2 log N + 3⌉ do
    for j ← 1, 2 do
      if i-th bit of (2sig + 1) is '1' then
        perform distance steps of the basic walk, starting with port
        firstPort;
        perform distance steps of the reverse basic walk, starting from the
        last port of entry;
      else
        remain idle for 2 · distance steps;
```

Lemma 3.2 *For any upper bound N, such that $N \geq n \geq N/16$, and $k \geq c \log N$, where c is a constant, a pair of agents equipped with k bits of memory, starting at non-symmetric initial positions with arbitrary delay, can achieve rendezvous in time $O(n^2/k + n \log n)$ using procedure* SmallMemoryRV.

Proof. Let v_1 and v_2 be the starting positions of the agents, and suppose that the agent starting from v_2 begins its first execution of procedure Oscillate not earlier than the agent starting from v_1. We show that rendezvous will be reached while the agents are performing procedure SmallMemoryRV, during the execution of the first call to subroutine Oscillate by the agent starting from v_2.

Let $\lambda(v)$ be a string of $\lceil 2 \log N + 3 \rceil$ bits $\{0, 1\}$, encoding the binary representation of the integer $2sig(v) + 1$. Since $sig(v_1) \neq sig(v_2)$, we have $\lambda(v_1) \neq \lambda(v_2)$. Moreover, $\lambda(v)$ ends with a 1, and since $sig(v) < 2^{\log N + 1}$, $\lambda(v)$ begins with $\lceil \log N + 1 \rceil$ zeros. Thus, $\lambda(v)$ is the lexicographically smallest string among all its rotations. Consequently, the inequality $\lambda(v_1) \neq \lambda(v_2)$ implies that for any integer l, $rot_l(\lambda(v_1)) \neq \lambda(v_2)$.

Now, consider the value of indices i and j within procedure Oscillate for the earlier agent in the round t during which the later agent begins its first execution of the same procedure. If $j = 1$, we set $l = i$, and if $j = 2$, we set $l = (i \mod \lceil 2 \log N + 3 \rceil) + 1$. Since $rot_l(\lambda(v_1)) \neq \lambda(v_2)$, there must exist a position p such that the p-th symbol of $rot_l(\lambda(v_1))$ is different from the p-th symbol of $\lambda(v_2)$. Now consider the time interval $[t + 4(N-1)p, t + 4(N-1)(p+1))$, i.e., the period when the later agent is performing the p-th iteration of the outer loop of the first execution of procedure Oscillate. During this interval, there exists a subinterval of $2(N-1)$ rounds during which one of the agents (the one whose currently used bit of λ is equal to one) performs the complete basic walk on T, while the other (the one whose currently used bit of λ is equal to zero) is stationary at some node. Hence, the agents will meet at this node. Since the execution time of procedure FindSignature is bounded by $O(n^2/k)$, and the execution time of procedure Oscillate is bounded by $O(n \log n)$, the number of rounds before rendezvous, counting from the beginning of the execution by the agent starting from node v_2, is bounded by $O(n^2/k + n \log n)$. □

Rendezvous procedure for agents with large memory

Procedure LargeMemoryRV, that is used when $k > N/\log N$, applies a more time-efficient approach to rendezvous by restricting the meeting location of the agents either to a specific node of the tree, or to one of the endpoints of a specific edge. Since the memory of the agent may be sublinear compared to the order

of the tree T, we do not perform a structural (e.g., DFS-based) analysis of the entire tree to determine such a location. Instead, the agent attempts to determine a meeting point in the so called *trimmed tree* T', which is the port-labeled tree given by the following construction (provided for purposes of definition, only):

1. Initially, let $T' = T$.
2. *Trimming.* Let $z = \lceil 32N/k \rceil$. Remove from T' all edges e such that one of the connected components of tree $T \setminus \{e\}$ has less than z nodes. Remove from T' all isolated nodes. Notice that, since $N < 16n$ and $k > \log n$, we have, in particular, $z < (n-1)/3$ for n large enough.
3. *Path contraction.* Remove from T' all nodes of degree 2 by contracting each path passing through such nodes into a single edge of the tree, preserving the port labeling at all the remaining nodes (of degree 1 or 3).

The above definition bears some resemblance to structures used in the parallel contraction algorithm from [29].

We remark that T' is a non-empty tree with at most $k/16$ nodes (see Lemma 3.3), which are by definition also nodes of tree T. The meeting node of the agents in procedure LargeMemoryRV is selected as follows. If the trimmed tree has a central node v, then the agents will meet at v. Otherwise, the trimmed tree must have a central edge e which corresponds to a path $(v_0, v_1, \ldots, v_l)$ in T of some length $l \geq 1$. If l is even, then the agents meet at the node $v_{l/2}$. Otherwise, the agents meet at one of the endpoints of the edge $\{v_{\lfloor l/2 \rfloor}, v_{\lceil l/2 \rceil}\}$ of T. Observe that since T' is uniquely defined, the node or pair of nodes which will be selected for rendezvous is independent of the starting positions of the agents.

The procedure relies on two key subroutines which allow the agent to navigate in the tree T'.

- Procedure TrimmedTreeNeighborhood, when called at a node u, computes the set of port numbers at node u which correspond to edges remaining after the trimming phase in the definition of T', i.e., edges of T leading from u to a subtree of at least z nodes. Testing if a port p at u leads to a sufficiently large subtree is implemented by performing $2z$ steps of the basic walk on T starting with port p at node u, memorizing the current tree-distance of the agent from u throughout this traversal. If the agent returns to u before completion of the last step of the walk, then the subtree has less than z nodes, and port p is not included in the output of the procedure.

```
procedure TrimmedTreeNeighborhood ()  { at node u }
  O ← ∅;
  for outPort ∈ [0.. deg(u) − 1]  do
    perform 2z steps of the basic walk, starting with port outPort, memorizing all ports
    of the performed traversal;
    if the entire subtree rooted at u and containing edge with port outPort has not been
    explored then
      O ← O ∪ {outPort};
    move back to u by performing 2z rounds of the reverse basic walk;
  return O;
```

- Procedure TraverseCompressedPath, when called at a node $u \in T'$ with a single argument *nextPort* (describing a port number at u in T') moves the agent using port number *nextPort*, to its neighbor w in T', following a contracted path in T. The values returned by the procedure are the port number by which w was entered when coming from u, and the length of the path in T connecting u and w. An optional second argument passed to TraverseCompressedPath allows

the agent to move a specified number of steps along the path between u and w in T, e.g., in order to reach its center.

```
procedure TraverseCompressedPath (nextPort, [maxLength (defaults to N)])
  distance ← 0;
  repeat
    move along the edge with port nextPort;
    returnPort ← port by which new node is entered;
    O ← TrimmedTreeNeighborhood ();
    nextPort ← any element of set O \ {returnPort};
    distance ← distance + 1;
  until distance = maxLength or |O| ≠ 2;
  return (returnPort, distance);
```

Procedure LargeMemoryRV consists of the following phases. First, the agent follows the basic walk on T, starting from its initial position, until it encounters the first node which is identified as a leaf of tree T', by using procedure TrimmedTreeNeighborhood. Next, the agent performs a basic walk in tree T', using procedures TrimmedTreeNeighborhood and TraverseCompressedPath to discover node neighborhoods and to navigate along edges of T', respectively. A basic walk in T' is defined as in tree T, with the additional condition that an agent leaving a leaf follows the only available port, regardless of its port number. The agent memorizes the entire port number sequence BW' used during this basic walk in T' and, by keeping track of the T' tree-distance from the starting node, detects the completion of the tour of the entire tree T'. Using local computations on the sequence BW', the agent now identifies the location of the central node or the central edge of tree T', expressed by the number of steps of the basic walk on T' required to reach this location from its initial position. If T' has a central node, then the agent reaches it, and stops, waiting for the other agent to arrive there. Otherwise, if T' has a central edge e, the agent proceeds to it and identifies the length l of the corresponding path $(v_0, v_1, \ldots, v_l)$ in T using procedure TraverseCompressedPath. If l is even, the agent moves to node $v_{l/2}$ by applying once more procedure TraverseCompressedPath, and stops. Otherwise, the agent reaches node $v_{\lfloor l/2 \rfloor}$ and applies procedure Oscillate. This is equivalent to performing SmallMemoryRV, but restricted to the two-node subtree (edge) $\{v_{\lfloor l/2 \rfloor}, v_{\lceil l/2 \rceil}\}$ of T.

```
procedure LargeMemoryRV () { starting at v }
  sig ← FindSignature ();
  while |TrimmedTreeNeighborhood()| ≠ 1 do
    traverse one step of the basic walk on T;
  { perform the complete basic walk on the reduced tree T', using procedure
  TrimmedTreeNeighborhood to discover ports leading to neighbors in T' and Tra-
  verseCompressedPath to traverse edges of T' }
  BW' ← basic walk string for reduced tree T' starting from the current location of the
  agent;
  { using string BW', locally compute whether T' has a central node or a central edge,
  and determine its location }
  i ← distance along basic walk on T' to central node/edge of T';
  move for i steps of the basic walk on T';
  if T' has a central node then
    stop {at central node of T' }
  else {T' has a central edge, which has just been reached}
    (returnPort, l) ← traverse central edge of T' using TraverseCompressedPath;
    { move to the center of the path in T corresponding to central edge of T' }
    (port, ·) ← TraverseCompressedPath (returnPort, ⌈l/2⌉);
    if l is even then
      stop {in the middle of the central path of T' }
    else
      repeat
        Oscillate (sig, 1, port)
      until rendezvous;
```

Lemma 3.3 *The tree T' is non-empty and has less than $k/16$ nodes.*

Proof. To prove that T' is non-empty, consider the node v of T with the property that the largest of the (at most three) connected components of $T \setminus \{v\}$ has the minimum possible number of nodes. Let $n_1 \geq (n-1)/3$ be the size of the largest connected component T_1 of $T \setminus \{v\}$, and let v_1 be the neighbor of v in T_1. We have $n - n_1 \geq n_1$, since otherwise the largest connected component of $T \setminus \{v_1\}$ would have fewer than n_1 nodes, violating the choice of v. For the edge $e = \{v, v_1\}$ we obtain that two connected components of $T \setminus \{e\}$ have $n - n_1$ and n_1 nodes, respectively, where $n - n_1 \geq n_1 \geq (n-1)/3 > z$. Consequently, tree T' contains edge e, and thus is non-empty.

To bound the size of T', consider the set of leaves $L = \{u_1, \ldots, u_l\}$ of tree T'. For a leaf u_i, let e_i be the unique edge of T incident to u_i such that all other edges incident to u_i in T are removed in the trimming phase of the construction of T'. Let X_i be the set of nodes of the connected component of $T \setminus \{e_i\}$ which contains u_i. By the definition of the trimmed tree, we have $|X_i| \geq z$, and moreover all the sets X_i are pairwise disjoint. Hence, we have the inequality: $n \geq \sum_{u_i \in L} |X_i| \geq |L|z$, which implies $|L| \leq n/z = n/\lceil 32N/k \rceil \leq k/32$. Since tree T' contains only nodes of degrees 1 and 3, the number of its nodes is precisely $2|L| - 2$, which is less than $k/16$. □

Lemma 3.4 *For any upper bound N, such that $N \geq n \geq N/16$, and $k \geq cN/\log N$, where c is a constant, a pair of agents equipped with k bits of memory, starting at non-symmetric initial positions with arbitrary delay, achieves rendezvous in time $O(n^2/k)$, using procedure* LargeMemoryRV.

Proof. If the tree T is such that the trimmed tree T' has a central node or a central edge formed by a contraction of an even path, then both agents eventually stop at the same node. Otherwise, the agents

eventually reach the endpoints of the same edge of T. Let v_1 and v_2 be the starting positions of the agents, and suppose that the agent starting from v_2 begins its first execution of procedure Oscillate not earlier than the agent starting from v_1. Similarly as in the proof of Lemma 3.2, rendezvous will be reached during the execution of the first call to procedure Oscillate by the agent starting from v_2.

Next, observe that LargeMemoryRV can be performed by an agent having k bits of memory. Throughout the procedure the agent has to store its signature, its position along the basic walks in T and T', and a constant number of local variables, each of size $O(\log N)$. Moreover, procedure LargeMemoryRV requires the memorization of the sequence of ports appearing in the basic walk BW' on tree T'. Since the order of T' is less than $k/16$ by Lemma 3.3, the storage space required for the string BW' is less than $k/2$ bits. Further steps of the procedure, involving computations of the central node or the central edge of this tree, require storage of a DFS stack for this tree, which may use an additional $k/8$ bits. Finally, procedure TrimmedTreeNeighborhood also requires the memorization of a basic walk on a subtree of length at most $2(z-1) < 2\lceil 32N/k \rceil$, which uses at most $k/4$ bits for a sufficiently large constant c. Note that none of the procedures is called recursively, hence the overall memory usage is less than k bits.

Finally, we bound the time required by the agent starting at v_2 to complete procedure LargeMemoryRV. The construction of string BW' involves a basic walk traversal of tree T'. Whenever a node of T' is entered, procedure TrimmedTreeNeighborhood is called, which requires $O(z) = O(n/k)$ time; the number of such calls is bounded by the length of BW' which is $O(k)$. Moreover, when calling procedure TraverseCompressedPath, procedure TrimmedTreeNeighborhood is executed as a subroutine. In this context, the procedure may be called at most twice for each node of T, resulting in $O(n)$ calls with a duration of $O(n/k)$ time each. Thus, the computation of BW' takes $O(n^2/k)$ time. After the sequence BW' has been computed, the identification of the position of the central node (or the central edge) of T' requires no further moves of the agent. Reaching this location is possible by performing a part of the basic walk on T', using once again $O(n^2/k)$ time. If the agent stops, then rendezvous has been reached in $O(n^2/k)$ time. Otherwise, if it performs procedure Oscillate, then similarly as in the proof of Lemma 3.2 we obtain that rendezvous is reached within this agent's first call to procedure Oscillate, and thus within an additional $O(\log n)$ rounds. The overall number of rounds performed by the agent starting at v_2 is thus bounded by $O(n^2/k)$. □

By Lemmas 3.2 and 3.4, the following algorithm solves the rendezvous problem in time $O(n^2/k)$ for a known linear upper bound N on n and for trees of maximum degree 3.

Algorithm 1: Rendezvous for known linear upper bound $N \geq n$ and degree $\Delta \leq 3$.

```
if k > N/log N then
    LargeMemoryRV ();
else
    SmallMemoryRV ();
```

3.2 Trees of approximately known size and arbitrary degrees

To extend our approach to trees of arbitrary maximum degree, we consider the labeled tree T^* obtained from T by splitting each node v of degree $\deg(v) \geq 3$, and replacing it by a subpath consisting of $\deg(v)$ nodes, defined as follows:

- node v is replaced by nodes $\{v_1^*, v_2^*, \ldots, v_{\deg(v)}^*\}$,
- there is an edge $\{v_1^*, v_2^*\}$ with ports 0 at v_1^* and v_2^*,
- for $2 \leq i \leq \deg(v) - 1$, there is an edge $\{v_i^*, v_{i+1}^*\}$ with port number 2 at v_i^* and number 0 at v_{i+1}^*,

- additionally, for each edge of T between a port p ($0 \leq p < \deg(v)$) at node v and a port q at node u, we add an edge to T^* leaving node v_p^* by port 1 and either entering node u by port q (if $\deg(u)<3$) or node u_q^* by port 1 (if $\deg(u)\geq 3$).

An agent can simulate a walk on tree T^* using only moves on tree T and local computations. As long as the visited nodes of T are of degree less than 3, the port labelings and moves on T^* correspond precisely to those on T. Upon entering a node v of T of degree at least 3 by port q, the agent simulates arrival by port 1 at node v_q^* of tree T^*. While located at nodes v_i^*, the agent can easily compute the port of T^* by which it reached the current node, the degree of v_i^*, and the destinations of the edges of T^* incident to this node. For these computations, it only requires an additional $O(\log \Delta)$ bits of storage for two variables, describing the degree of the current node, and the index i of the agent's virtual location v_i^* in T^*. The values of these variables no longer need to be stored upon moving to another node of T.

Note that the transformation of tree T into tree T^* is a bijection. For a given tree T^*, the original tree T can be reconstructed by detecting all paths of the form $\{v_1^*, v_2^*, \ldots, v_{\deg(v)}^*\}$ in T^* (based on the arrangement of ports 0 and 2 at nodes of degree 3 in T^*), and then regaining the connections of the corresponding node v in T. Since the bijection between T and T^* clearly maps symmetric trees into symmetric trees, it follows that if rendezvous is feasible for an initial configuration of agents in T, then it is feasible for the corresponding configuration in T^*.

Rendezvous in tree T may be achieved by applying all the procedures from the previous subsection, and replacing each step of the basic walk (or reverse basic walk) on T by a corresponding step on T^*. Indeed, a virtual rendezvous in T^* is a sufficient condition for rendezvous in T: agents which in their simulation arrive at the same location v_i^* in tree T^* will in fact meet at node v of tree T.

In all the computations, the agent needs to assume an upper bound N^* on the order of T^* in place of the upper bound N. Since each node v of T is replaced by at most $\deg(v)$ nodes of T^*, the number of nodes of T^* is bounded by $\sum_{v \in V}(\deg(v)) < 2n$. Hence, we can put $N^* = 2N$, and so all the asymptotic complexity results shown for trees of degree at most 3 are preserved in the case of trees with arbitrary maximum degree.

The above modification applies also when no upper bound on the order of the tree is known in advance, hence in the next subsection we may assume that the maximum degree of the tree is at most 3.

3.3 The general algorithm

The algorithm for rendezvous in trees with no known upper bound $N \geq n$ is a refinement of Algorithm 1. The agent attempts to iterate a variant of Algorithm 1 for growing assumed values of N, starting from $N = k$, and increasing them by a multiplicative factor of 4 in subsequent iterations.

Within the range of values $N \geq k \geq N/\log N$, the agent attempts to execute procedure LargeMemoryRV, checking if the claim of Lemma 3.3 is not violated, i.e., if the trimmed tree T' has at most $k/16$ nodes. If this is the case, then the agent reaches rendezvous within the current call to procedure LargeMemoryRV, in view of Lemma 3.4. Otherwise, the agent returns to its initial position, and knowing the previously assumed upper bound of N to be insufficient, repeats the process for a 4 times larger value of N.

As soon as the condition $k < N/\log N$ is fulfilled, the agent applies a different approach, which is a variant of procedure SmallMemoryRV. In each iteration, for the assumed value of N, the agent first performs a synchronizing block of $8(N-1)$ rounds, consisting of $2(N-1)$ rounds of waiting, followed by $2(N-1)$ steps of the basic walk, another $2(N-1)$ rounds of waiting, and $2(N-1)$ steps of the reverse basic walk. This is intended to facilitate rendezvous if the other agent has already begun execution for a larger upper bound on n. If the agents do not meet during this block, the agent computes its signature using procedure FindSignature, and synchronizes by waiting until the completion time of the slowest possible execution of FindSignature for the given value of N. Finally, the agent repeatedly executes

procedure OSCILLATE, so that the number of rounds spent within procedure OSCILLATE is at least equal to the number of rounds used when computing the agent's signature. If rendezvous is not achieved during these repetitions of procedure OSCILLATE, the entire process is repeated for a 4 times larger value of N. We will show that the algorithm will, at the very latest, reach rendezvous in the $(m+1)$-st repetition of the process, where m is the earliest repetition in which the value of N it assumes is larger than n. Consequently, the assumption $N < 16n$ is valid throughout the algorithm.

Algorithm 2: Rendezvous in arbitrary trees.

```
N ← k;
repeat { Phase 1: attempt rendezvous with large memory }
  try LARGEMEMORYRV (), aborting if tree T' has more than k/16 nodes;
  { the further steps are executed only if the assumed bound is too small
  (N < n) }
  return to the starting position;
  N ← 4N;
until k < N/ log N;
repeat { Phase 2: achieve rendezvous with small memory }
  t ← current round number;
  wait for 2(N − 1) rounds; { (a) }
  perform 2(N − 1) steps of the basic walk; { (b) }
  wait for 2(N − 1) rounds; { (c) }
  perform 2(N − 1) steps of the reverse basic walk; { (d) }
  τ ← 8(N − 1) + duration of the slowest possible execution of FINDSIG-
  NATURE for current N;
  sig ← FINDSIGNATURE ();
  wait until round number t + τ;
  repeat
    OSCILLATE (sig, 2(N − 1), 0);
  until round number is larger than t + 2τ;
  N ← 4N;
until rendezvous;
```

Theorem 3.1 *For any $k \geq c \log n$, where c is a constant, a pair of agents equipped with k bits of memory, starting at non-symmetric initial positions with arbitrary delay, achieves rendezvous in time $O(n^2/k)$ using Algorithm 2.*

Proof. Let $N_i = 4^i k$, and let l be the smallest non-negative integer such that the trimmed tree T' for parameter $N = N_l$ has at most $k/16$ nodes. The proof is split into two cases.

Case 1 ($k \geq N_l / \log N_l$). In this case, the agents successfully execute procedure LARGEMEMORYRV for $N = N_l$ without aborting (using at most k bits of memory), and either stop at the same node, or meet at one of the endpoints of the same edge of T, achieving rendezvous. To bound the time of execution, observe that since the tree T' for $N = N_{l-1} = N_l/4$ violates the claim of Lemma 3.3, we must have $N_l/4 < n$. Since the execution time of the i-th iteration of Phase 1 of Algorithm 2 is bounded by $O(N_i^2/k) = O(4^{2i}k)$, the overall execution time of the first $l+1$ iterations of the loop is $O(4^{2l}k)$. Since $n > N_l/4 = 4^{l-1}k$, rendezvous is reached in $O(n^2/k)$ time.

Case 2 ($k < N_l / \log N_l$). In this case, both agents proceed to execute Phase 2 of Algorithm 2 after $O(n^2/k)$ rounds. Suppose w.l.o.g. that agent A_2 starts the execution of Phase 2 exactly $\theta' \geq 0$ rounds after agent A_1. Since the duration of any iteration of the loop is the same for both agents, agent A_1 will start the

i-th iteration in some round t_i, while agent A_2 will start it in round $t_i + \theta'$. Let N'_i denote the value of N used by the agent in the i-th iteration of this loop. Observe that Phase 2 of the algorithm and all its subroutines are constructed so that throughout each interval of $2(N'_i - 1)$ consecutive rounds of the form $[t_i + 2(N'_i - 1)a, t_i + 2(N'_i - 1)(a + 1)) \subset [t_i, t_{i+1})$, where a is an integer, agent A_1 always performs one of the following three actions: it either remains motionless for $2(N'_i - 1)$ rounds, or performs $2(N'_i - 1)$ steps of the basic walk on T, or performs $2(N'_i - 1)$ steps of the reverse basic walk.

Let m be the smallest integer such that $N'_m \geq n$. Consider the round $t_m + \theta'$ when agent A_2 starts the m-th iteration of the loop. Suppose that $t_m + \theta' \geq t_{m+1}$, i.e., agent A_1 has already started executing (at least) the $(m + 1)$-st iteration of the loop at this time. Then, rendezvous will be achieved while agent A_2 is executing one of the lines (a), (b), (c), or (d) in its m-th iteration of the loop. To show this, we consider the following possibilities:

- Throughout all rounds in the interval $[t_m + \theta', t_m + \theta' + 4(N'_m - 1))$, agent A_1 is performing steps of the basic walk (resp., of the reverse basic walk). Then, during the time interval $[t_m + \theta', t_m + \theta' + 2(N'_m - 1))$, this agent visits all the nodes of the tree at least once, since $2(N'_m - 1) \geq 2(n - 1)$, which is the length of a single traversal of the tree using the basic walk. Hence, agent A_1 will meet agent A_2, which remains stationary throughout the considered time interval (line (a)).

- Throughout all rounds in the interval $[t_m + \theta', t_m + \theta' + 4(N'_m - 1))$, agent A_1 remains motionless. During the time interval $[t_m + \theta' + 2(N'_m - 1), t_m + \theta' + 4(N'_m - 1))$, agent A_2 visits all the nodes of the tree at least once (while performing the basic walk in line (b)), hence it meets the stationary agent A_1, achieving rendezvous.

- Agent A_1 performs at least two different actions during the interval $[t_m + \theta', t_m + \theta' + 4(N'_m - 1))$. Then, the agent will always perform the same action during the time interval $[t_m + \theta' + 4(N'_m - 1), t_m + \theta' + 8(N'_m - 1))$, since it can change the type of performed action after $2(N'_{m+1} - 1) > 8(N'_m - 1)$ steps at the earliest, by the construction of the algorithm. Depending on the type of action performed by agent A_1 in this time interval, it will meet agent A_2 while A_2 is performing either line (c) or line (d) of its m-th iteration of the loop; the details of the proof are the same as in the two cases above.

Thus, if the agents do not meet in the m-th iteration of the loop by A_1, then $t_m + \theta' < t_{m+1}$. From this, in view of $t_{i+1} - t_i \in \Theta(N'^2_i/k)$, we obtain: $\theta' < t_{m+1} - t_m < (t_{m+2} - t_{m+1})/4$. By the construction of the algorithm, in the $(m + 1)$-st iteration of the loop agent A_2 performs its first call to procedure Oscillate exactly θ' rounds after agent A_1, and moreover the total duration of the calls to procedure Oscillate in this $(m + 1)$-st iteration of the loop is at least $(t_{m+2} - t_{m+1})/2$. Consequently, for $(t_{m+2} - t_{m+1})/2 - \theta' > (t_{m+2} - t_{m+1})/4$ rounds, the agents are concurrently repeating executions of procedure Oscillate with parameter $N = N'_{m+1} > n$. Since the execution time of procedure Oscillate is $O(N'_{m+1} \log N'_{m+1})$, this number of rounds is sufficient for agent A_2 to complete its first execution of procedure Oscillate. By the proof of Lemma 3.2, rendezvous will thus be achieved by the end of the first execution of procedure Oscillate by agent A_2.

To bound the time until rendezvous, note that the duration of the i-th loop of Phase 2 is $\Theta(N'^2_i/k)$ rounds, and moreover $N'_{m-1} < n$, thus $N'_{m+1} = 16N'_{m-1} < 16n$. Hence, the number of rounds required by agent A_2 to complete Phase 2 is $O(n^2/k)$. Since the same bound holds also for Phase 1, the entire algorithm is completed within $O(n^2/k)$ rounds.

Since the implementation of Algorithm 2 relies on calls to subroutines which are never recursive, the minimum amount of memory k for which the approach works correctly can be upper-bounded as follows. Whenever an execution path of the algorithm is chosen for which procedure SmallMemoryRV is called, we assume that the compared views are of length 1. Now, by summing the maximum possible sizes (in binary representation) of all the named variables in Algorithm 2 and its subroutines, and adding a constant number of bits of memory to represent the procedure call stack and the number of the currently executed

line in the current procedure, we obtain an upper bound on the amount of memory which is indispensable for the algorithm to function correctly. As a rough estimate, it is sufficient to require that $k \geq c \log n$, where $c = 100$. □

We remark that our Algorithm 2 works for all non-symmetric starting configurations, i.e., configurations for which rendezvous is always feasible. It turns out that when the configuration is symmetric, rendezvous is not feasible when the agents start simultaneously, but becomes feasible when one of the agents starts with some non-zero delay [18]. The question of whether our time-space trade-off holds for the case of symmetric positions with non-zero delay remains open.

4 Lower bounds

In this section we establish two lower bounds. The first is on the size of memory needed for rendezvous with simultaneous start, and the second is on rendezvous time with given memory. Theorem 4.1 from [21] says that rendezvous with simultaneous start from arbitrary non-symmetric initial positions in the class of trees with at most n nodes and at most l leaves is possible with agents equipped with $O(\log l + \log\log n)$ memory bits. In particular, it implies that rendezvous with simultaneous start from arbitrary non-symmetric initial positions in a n-node line can be achieved with $O(\log\log n)$ memory bits. Our first lower bound shows that this is not true and that the assumption of our rendezvous algorithm that the number of memory bits is at least logarithmic, cannot be removed, even for the class of lines. Our second lower bound concerns the trade-off between memory size and time of rendezvous. Again, it holds even for simultaneous start and even in the class of lines of known length, and shows that the time of our rendezvous algorithm is the best possible for any memory size for which rendezvous is feasible. Since a part of the proofs of both lower bounds is the same, we state them as one theorem.

Theorem 4.1 *Consider a pair of agents equipped with k bits of memory and achieving rendezvous in any n-node line starting from arbitrary non-symmetric initial positions. Then:*

1. *For some constant c_1 and arbitrarily large n, we have $k \geq c_1 \log n$.*
2. *For some constant c_2 and arbitrarily large n, there exists a n-node line for which these agents use time at least $c_2(n + n^2/k)$ to accomplish rendezvous from some non-symmetric initial positions, even for simultaneous start.*

Proof. Consider a n node line for even n. Let L be the part (segment) of the line with $\lceil n/3 \rceil + 1$ nodes starting from one end, let R be the part of the line with $\lceil n/3 \rceil + 1$ nodes starting from the other end, and let M be the remaining middle part of the line. Since ports at every node of degree 2 can be numbered in two different ways, there are at least $2^{n/3}$ possible port numberings for part L and for part R. The number of edges in part M is odd and it is at least $n/4$, for sufficiently large n. Fix the following port numbering of M: the central edge of M has ports 0 at both ends and all other edges in M have the same port numbers at both ends. Let u be the extremity of M adjacent to L and let v be the extremity of M adjacent to R. Let u' be the node in L adjacent to u and let v' be the node in R adjacent to v. Agents start simultaneously from initial positions u and v. Let $K = 2^k$ be the number of memory states of an agent. Let τ be the maximum rendezvous time for any rendezvous algorithm in such a line. Then $\tau \leq (nK)^2$. Indeed, if the agents do not meet after time $(nK)^2$, then by the pigeonhole principle there exist two rounds $t_1 < t_2 \leq (nK)^2 + 1$ such that the states and locations of both agents in round t_1 and in round t_2 are identical. From round t_2 on, the pair of agents must repeat infinitely the same loop, precluding the possibility of rendezvous.

Proof of Part 1. In this part of the proof we use an argument similar to that in [21] (Theorem 4.1). Fix an agent with the set S of states and fix a part L or R of the line. Call this part P. (P is

treated as a sequence of ports starting from the respective endpoint of the line.) We define the function $q : S \to S \times \{1, \ldots, \tau\}$, called the *behavior function*, by the formula $q(s) = (s', t)$, where s' is the state in which the agent entering part P (by node u' or v') in state s leaves this part, and t is the number of rounds to complete the visit of part P when starting in state s. The number of possible behavior functions is at most $F = (K\tau)^K$. A behavior function depends only on the part P for which it is constructed. Assume $k < \frac{1}{3}\log n$. For sufficiently large k we have:

$$\log K + \log\log(K\tau) \leq k + \log\log(n^2K^3) \leq k + \log(2\log n + 3k) < k + \log\log n + \log k + 3 < \frac{2}{3}\log n.$$

Hence $K\log(K\tau) < n^{2/3} < n/3$, and consequently $F = (K\tau)^K < 2^{n/3}$. Thus the number of possible behavior functions is strictly smaller than the total number of possible parts P. It follows that there are two such parts P_1 and P_2 for which the corresponding behavior functions are equal.

Consider two instances of the rendezvous problem in a n-node line. Within part M, both of them have the port labeling defined above. One instance has both parts L and R equal to P_1 and the second instance has $L = P_1$ and $R = P_2$. (In each case the sequence of ports of P_i has to be inserted starting from the endpoint of the line.) Rendezvous is impossible in the first instance because in this instance initial positions of the agents form a symmetric pair of nodes. Consider the second instance, in which the initial positions of the agents do not form a symmetric pair. Because of the symmetry of the port numbering of the part M, agents cannot meet inside any of the side parts. Indeed, when one of them is in L, the other one is in R. Since the behavior function associated with parts P_1 and P_2 is the same, the agents leave these parts always at the same time and in the same state. Hence they cannot meet in the part M, in view of the symmetry of their positions and states with respect to the central edge of M. This implies that they never meet, in spite of asymmetric initial positions. Hence rendezvous in the second instance requires at least $\frac{1}{3}\log n$ bits of memory.

Proof of Part 2. Fix an agent with the set S of states and fix a symmetric n-node line $\mathcal{L}$, i.e., a line with identical parts L and R. For $s \in S$ and for any round t, we say that the agent is in *configuration* (s, t) in node w, if it leaves node w in state s at time t. We define a *long trip* of the agent as a sequence of moves starting at an extremity of M, traversing only edges of M, traversing the central edge at least once, and ending at an extremity of M. The duration of a long trip is at least $n/4$. A *critical configuration* is a configuration of the agent at the beginning of a long trip. A *sequence of critical configurations* (SCC) is a sequence of consecutive critical configurations starting at time 0 at the initial position of the agent.

We may assume that $k \geq \frac{1}{3}\log n$; otherwise rendezvous is impossible by the proof of Part 1. Assume that $x = \lfloor cn/k \rfloor$, where $c = \frac{1}{28}$. Since there exist at most $K\tau$ different configurations, where $\tau \leq (nK)^2$, the number $(K\tau)^x$ of all SCC's of length x, taken over all symmetric n-node lines, is at most $(n^2K^3)^x$. We have

$$x\log(K\tau) \leq x\log(n^2K^3) \leq \frac{cn}{k}\log(n^2K^3) = \frac{cn}{k}(2\log n + 3k) \leq \frac{cn}{k}(9k) = 9cn < \frac{n}{3}.$$

Hence $(K\tau)^x < 2^{n/3}$. By the pigeonhole principle it follows that there exist at least two distinct symmetric lines $\mathcal{L}_1$ and $\mathcal{L}_2$ whose both side parts are, respectively, P_1 and P_2, for which the SCC of length x is the same. Let $(\gamma_1, \ldots, \gamma_x)$ be this common SCC.

Now consider the n-node line $\mathcal{L}_3$, for which $L = P_1$ and $R = P_2$. This line is not symmetric, hence the initial positions u and v of the agents in $\mathcal{L}_3$ are not a symmetric pair. We show that rendezvous of the agents cannot happen before each of them accomplishes at least x long trips. Call the agent starting at u the *left* agent, and the agent starting at v the *right* agent. The behavior of the left agent before the beginning of the first long trip is the same as that of the left agent in $\mathcal{L}_1$ and the behavior of the right agent before the beginning of the first long trip is the same as that of the right agent in $\mathcal{L}_2$. Hence both agents start their first long trip in configuration γ_1. Similarly, by induction on the long trip number, the agents start the i-th long trip in configuration γ_i, for $i \leq x$. During the periods between long trips, the agents are on different sides of the central edge, hence they cannot meet. During the long trips they cannot meet

either, because these trips are executed inside the part M which is a symmetric line and configurations of agents at the beginning of each such trip are identical.

This shows that rendezvous cannot occur before each agent accomplishes at least x long trips. Since each long trip has duration at least $n/4$, the rendezvous time is at least $xn/4 \geq \frac{cn}{2k} \cdot \frac{n}{4} = \frac{1}{224}\frac{n^2}{k}$. The linear lower bound on rendezvous time is obvious, as the distance between initial positions of the agents is at least $n/4$. This completes the proof. □

We remark that the lower bounds we present hold for deterministic agents. It would be interesting to obtain lower-bounds on the expected rendezvous time in trees for randomized agents, possibly in a model similar to that considered for the ring in [26].

References

[1] S. Alpern, The rendezvous search problem, SIAM J. on Control and Optimization 33 (1995), 673-683.

[2] S. Alpern, Rendezvous search on labelled networks, Naval Reaserch Logistics 49 (2002), 256-274.

[3] S. Alpern and S. Gal, The theory of search games and rendezvous. Int. Series in Operations research and Management Science, Kluwer Academic Publisher, 2002.

[4] J. Alpern, V. Baston, and S. Essegaier, Rendezvous search on a graph, Journal of Applied Probability 36 (1999), 223-231.

[5] C. Ambühl, L. Gasieniec, A. Pelc, T. Radzik, X. Zhang, Tree exploration with logarithmic memory, ACM Transactions on Algorithms 7:2 (2011), article 17.

[6] E. Anderson and R. Weber, The rendezvous problem on discrete locations, Journal of Applied Probability 28 (1990), 839-851.

[7] E. Anderson and S. Fekete, Asymmetric rendezvous on the plane, Proc. 14th Annual ACM Symp. on Computational Geometry (1998), 365-373.

[8] E. Anderson and S. Fekete, Two-dimensional rendezvous search, Operations Research 49 (2001), 107-118.

[9] D. Baba, T. Izumi, F. Ooshita, H. Kakugawa, T. Masuzawa, Space-optimal rendezvous of mobile agents in asynchronous trees. Proc. 17th Int. Colloquium on Structural Information and Comm. Complexity, (SIROCCO 2010), LNCS 6058, 86-100.

[10] V. Baston and S. Gal, Rendezvous on the line when the players' initial distance is given by an unknown probability distribution, SIAM J. on Control and Opt. 36 (1998), 1880-1889.

[11] V. Baston and S. Gal, Rendezvous search when marks are left at the starting points, Naval Reaserch Logistics 48 (2001), 722-731.

[12] M. Cieliebak, P. Flocchini, G. Prencipe, N. Santoro, Solving the robots gathering problem, Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP 2003), 1181-1196.

[13] J. Czyzowicz, A. Kosowski, A. Pelc, How to meet when you forget: Log-space rendezvous in arbitrary graphs, Distributed Computing 25 (2012), 165-178.

[14] J. Czyzowicz, A. Labourel, A. Pelc, How to meet asynchronously (almost) everywhere, ACM Transactions on Algorithms 8 (2012), article 37.

[15] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, U. Vaccaro, Asynchronous deterministic rendezvous in graphs, Theoretical Computer Science 355 (2006), 315-326.

[16] A. Dessmark, P. Fraigniaud, D. Kowalski, A. Pelc. Deterministic rendezvous in graphs. Algorithmica 46 (2006), 69-96.

[17] J.P. Duval, Factorizing words over an ordered alphabet, Journal of Algorithms 4 (1983), 363-381.

[18] S. Elouasbi, A. Pelc, Time of anonymous rendezvous in trees: Determinism vs. randomization, Proc. 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2012), LNCS 7355, 291-302.

[19] P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous robots with limited visibility, Theoretical Computer Science 337 (2005), 147-168.

[20] P. Fraigniaud, A. Pelc, Deterministic rendezvous in trees with little memory, Proc. 22nd International Symposium on Distributed Computing (DISC 2008), LNCS 5218, 242-256.

[21] P. Fraigniaud, A. Pelc, Delays induce an exponential memory gap for rendezvous in trees, Proc. 22nd Ann. ACM Symposium on Parallel Algorithms and Architectures (SPAA 2010), 224-232.

[22] P. Fraigniaud, A. Pelc, Delays induce an exponential memory gap for rendezvous in trees, ACM Transactions on Algorithms 9:2 (2013), article 17.

[23] S. Gal, Rendezvous search on the line, Operations Research 47 (1999), 974-976.

[24] A. Israeli and M. Jalfon, Token management schemes and random walks yield self stabilizing mutual exclusion, Proc. 9th Annual ACM Symposium on Principles of Distributed Computing (PODC 1990), 119-131.

[25] D. Kowalski, A. Malinowski, How to meet in anonymous network, Theoretical Computer Science 399 (2008), 141-156.

[26] E. Kranakis, D. Krizanc, and P. Morin, Randomized rendezvous with limited memory, ACM Transactions on Algorithms, 7:3 (2011), article 34.

[27] E. Kranakis, D. Krizanc, N. Santoro and C. Sawchuk, Mobile agent rendezvous in a ring, Proc. 23rd Int. Conference on Distributed Computing Systems (ICDCS 2003), IEEE, 592-599.

[28] W. Lim and S. Alpern, Minimax rendezvous on the line, SIAM J. on Control and Optimization 34 (1996), 1650-1665.

[29] G. Miller, J. Reif, Parallel Tree Contraction, Part 1: Fundamentals. Randomness and Computation (1989), JAI Press, 47-72.

[30] A. Pelc, Deterministic rendezvous in networks: A comprehensive survey, Networks 59 (2012), 331-347.

[31] K. Roh, M. Crochemore, C.S. Iliopoulos, K. Park, External Memory Algorithms for String Problems, Fundamenta Informaticae 84 (2008), 17-32.

[32] A. Ta-Shma and U. Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), 599-608.

[33] L. Thomas, Finding your kids when they are lost, Journal on Operational Res. Soc. 43 (1992), 637-639.

[34] X. Yu and M. Yung, Agent rendezvous: a dynamic symmetry-breaking problem, Proc. International Colloquium on Automata, Languages, and Programming (ICALP 1996), LNCS 1099, 610-621.